Software quality management[*]

Hung Vo

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License [†]

Abstract

This module emphasizes software quality engineering as an integral facet of development, from re-
quirements through delivery and maintenance. We will understand how to choose appropriate quality
goals and select, plan, and execute quality assurance activities throughout development and evolution to
predictably meet quality and schedule goals. We introduce also how quality assurance can be incorporated
into process improvement feedback loops that amplify the ability of an organization to cost-effectively
prevent and detect faults.

1 Introduction

What is software quality, and why is it so important that it is pervasive in the Software Engineering Body
of Knowledge? Within an information system, software is a tool, and tools have to be selected for quality
and for appropriateness. That is the role of equirements. But software is more than a tool. It dictates the
performance of the system, and it is therefore important to the system quality.

The notion of "quality" is not as simple as it may seem. For any engineered product, there are many
desired qualities relevant to a particular project, to be discussed and determined at the time that the product
requirements are determined. Qualities may be present or absent, or may be matters of degree, with tradeoffs
among them, with practicality and cost as major considerations. The software engineer has a responsibility to
elicit the system's quality requirements that may not be explicit at the outset and to discuss their importance
and the difficulty of attaining them. All processes associated with software quality (e.g. building, checking,
improving quality) will be designed with these in mind

and carry costs based on the design. Thus, it is important to have in mind some of the possible attributes
of quality.

Various researchers have produced models (usually taxonomic) of software quality characteristics or
attributes that can be useful for discussing, planning, and rating the quality of software products. The
models often include metrics to "measure" the degree of each quality attribute the product attains.

Usually these metrics may be applied at any of the product levels. They are not always direct measures of
the quality characteristics of the finished product, but may be relevant to the achievement of overall quality.
Each model may have a different set of attributes at the highest level of the taxonomy, and selection of
and definitions for the attributes at all levels may differ. The important point is that the system software
requirements define the quality requirements and the definitions of the attributes for them.

---

2 Software Quality Fundamentals

Agreement on quality requirements, as well as clear communication to the software engineer on what constitutes quality, require that the many aspects of quality be formally defined and discussed.

A software engineer should understand the underlying meanings of quality concepts and characteristics and their value to the software under development or to maintenance.

The important concept is that the software requirements define the required quality characteristics of the software and influence the measurement methods and acceptance criteria for assessing these characteristics.

2.1 Software Engineering Culture and Ethics

Software engineers are expected to share a commitment to software quality as part of their culture.

Ethics can play a significant role in software quality, the culture, and the attitudes of software engineers. The IEEE Computer Society and the ACM have developed a code of ethics and professional practice based on eight principles to help software engineers reinforce attitudes related to quality and to the independence of their work.

2.2 Value and Costs of Quality

The notion of "quality" is not as simple as it may seem. For any engineered product, there are many desired qualities relevant to a particular perspective of the product, to be discussed and determined at the time that the product requirements are set down. Quality characteristics may be required or not, or may be required to a greater or lesser degree, and trade-offs may be made among them.

The cost of quality can be differentiated into prevention cost, appraisal cost, internal failure cost, and external failure cost.

A motivation behind a software project is the desire to create software that has value, and this value may or may not be quantified as a cost. The customer will have some maximum cost in mind, in return for which it is expected that the basic purpose of the software will be fulfilled. The customer may also have some expectation as to the quality of the software. Sometimes customers may not have thought through the quality issues or their related costs. Is the characteristic merely decorative, or is it essential to the software? If the answer lies somewhere in between, as is almost always the case, it is a matter of making the customer a part of the decision process and fully aware of both costs and benefits. Ideally, most of these decisions will be made in the software requirements process, but these issues may arise throughout the software life cycle. There is no definite rule as to how these decisions should be made, but the software engineer should be able to present quality alternatives and their costs.

2.3 Models and Quality Characteristics

Terminology for software quality characteristics differs from one taxonomy (or model of software quality) to another, each model perhaps having a different number of hierarchical levels and a different total number of characteristics. Various authors have produced models of software quality characteristics or attributes which can be useful for discussing, planning, and rating the quality of software products. ISO/IEC has defined three related models of software product quality (internal quality, external quality, and quality in use) (ISO9126-01) and a set of related parts (ISO14598-98).

2.3.1 Software engineering process quality

Software quality management and software engineering process quality have a direct bearing on the quality of the software product.

Models and criteria which evaluate the capabilities of software organizations are primarily project organization and management considerations, and, as such, are covered in the Software Engineering Management and Software Engineering Process.

Of course, it is not possible to completely distinguish the quality of the process from the quality of the product.

Process quality influences the quality characteristics of software products, which in turn affect quality-in-use as perceived by the customer.

Two important quality standards are TickIT and one which has an impact on software quality, the ISO9001-00 standard, along with its guidelines for application to software.

Another industry standard on software quality is CMMI. CMMI intends to provide guidance for improving processes. Specific process areas related to quality management are process and product quality assurance, process verification, and process validation. CMMI classifies reviews and audits as methods of verification, and not as specific processes like.

There was initially some debate over whether ISO9001 or CMMI should be used by software engineers to ensure quality. This debate is widely published, and, as a result, the position has been taken that the two are complementary and that having ISO9001 certification can help greatly in achieving the higher maturity levels of the CMMI.

2.3.2 Software product quality

The software engineer needs, first of all, to determine the real purpose of the software. In this regard, it is of prime importance to keep in mind that the customer's requirements come first and that they include quality requirements, not just functional requirements. Thus, the software engineer has a responsibility to elicit quality requirements which may not be explicit at the outset and to discuss their importance as well as the level of difficulty in attaining them. All processes associated with software quality (for example, building, checking, and improving quality) will be designed with these requirements in mind, and they carry additional costs.

Standard ISO9126-01 defines, for two of its three models of quality, the related quality characteristics and sub-characteristics, and measures which are useful for assessing software product quality.

The meaning of the term "product" is extended to include any artifact which is the output of any process used to build the final software product. Examples of a product include, but are not limited to, an entire system requirements specification, a software requirements specification for a software component of a system, a design module, code, test documentation, or reports produced as a result of quality analysis tasks. While most treatments of quality are described in terms of the final software and system performance, sound engineering practice requires that intermediate products relevant to quality be evaluated throughout the software engineering process.

2.4 Quality Improvement

The quality of software products can be improved through an iterative process of continuous improvement which requires management control, coordination, and feedback from many concurrent processes: the software life cycle processes; the process of error/defect detection, removal, and prevention; and the quality improvement process.

The theory and concepts behind quality improvement, such as building in quality through the prevention and early detection of errors, continuous improvement, and customer focus, are pertinent to software engineering. These concepts are based on the work of experts in quality who have stated that the quality of a product is directly linked to the quality of the process used to create it.

Approaches such as the Total Quality Management (TQM) process of Plan, Do, Check, and Act (PDCA) are tools by which quality objectives can be met. Management sponsorship supports process and product evaluations and the resulting findings. Then, an improvement program is developed identifying detailed actions and improvement projects to be addressed in a feasible time frame. Management support implies that each improvement project has enough resources to achieve the goal defined for it. Management sponsorship must be solicited frequently by implementing proactive communication activities. The involvement of work groups, as well as middle-management support and resources allocated at project level.

3 Software Quality Management Processes

Software quality management (SQM) applies to all perspectives of software processes, products, and re-
sources. It defines processes, process owners, and requirements for those processes, measurements of the
process and its outputs, and feedback channels. Software quality management processes consist of many
activities. Some may find defects directly, while others indicate where further examination may be valuable.
The latter are also referred to as direct-defect-finding activities. Many activities often serve as both.

Planning for software quality involves:

- Defining the required product in terms of its quality characteristics.
- Planning the processes to achieve the required product.

These aspects differ from, for instance, the planning SQM processes themselves, which assess planned quality
characteristics versus actual implementation of those plans. The software quality management processes
must address how well software products will, or do, satisfy customer and stakeholder requirements, provide
value to the customers and other stakeholders, and provide the software quality needed to meet software
requirements.

SQM can be used to evaluate the intermediate products as well as the final product.

Some of the specific SQM processes are defined in standard (IEEE12207.0-96):

- Quality assurance process
- Verification process
- Validation process
- Review process
- Audit process

These processes encourage quality and also find possible problems. But they differ somewhat in their
emphasis.

SQM processes help ensure better software quality in a given project. They also provide, as a by-product,
general information to management, including an indication of the quality of the entire software engineering
process. The Software Engineering Process and Software Engineering Management KAs discuss quality
programs for the organization developing the software. SQM can provide relevant feedback for these areas.

SQM processes consist of tasks and techniques to indicate how software plans (for example, management,
development, configuration management) are being implemented and how well the intermediate and final
products are meeting their specified requirements. Results from these tasks are assembled in reports for
management before corrective action is taken. The management of an SQM process is tasked with ensuring
that the results of these reports are accurate.

As described in this KA, SQM processes are closely related; they can overlap and are sometimes even
combined. They seem largely reactive in nature because they address the processes as practiced and the
products as produced; but they have a major role at the planning stage in being proactive in terms of the
processes and procedures needed to attain the quality characteristics and degree of quality needed by the
stakeholders in the software.

Risk management can also play an important role in delivering quality software. Incorporating disciplined
risk analysis and management techniques into the software life cycle processes can increase the potential for
producing a quality product.

3.1 Software Quality Assurance

SQA processes provide assurance that the software products and processes in the project life cycle conform
to their specified requirements by planning, enacting, and performing a set of activities to provide adequate
confidence that quality is being built into the software. This means ensuring that the problem is clearly
and adequately stated and that the solution's requirements are properly defined and expressed. SQA seeks
to maintain the quality throughout the development and maintenance of the product by the execution of a

variety of activities at each stage which can result in early identification of problems, an almost inevitable feature of any complex activity. The role of SQA with respect to process is to ensure that planned processes are appropriate and later implemented according to plan, and that relevant measurement processes are provided to the appropriate organization.

The SQA plan defines the means that will be used to ensure that software developed for a specific product satisfies the user's requirements and is of the highest quality possible within project constraints. In order to do so, it must first ensure that the quality target is clearly defined and understood. It must consider management, development, and maintenance plans for the software. Refer to standard (IEEE730-98) for details.

The specific quality activities and tasks are laid out, with their costs and resource requirements, their overall management objectives, and their schedule in relation to those objectives in the software engineering management, development, or maintenance plans. The SQA plan should be consistent with the software configuration management plan. The SQA plan identifies documents, standards, practices, and conventions governing the project and how they will be checked and monitored to ensure adequacy and compliance. The SQA plan also identifies measures, statistical techniques, procedures for problem reporting and corrective action, resources such as tools, techniques, and methodologies, security for physical media, training, and SQA reporting and documentation. Moreover, the SQA plan addresses the software quality assurance activities of any other type of activity described in the software plans, such as procurement of supplier software to the project or commercial off-the-shelf software (COTS) installation, and service after delivery of the software. It can also contain acceptance criteria as well as reporting and management activities which are critical to software quality.

3.2 Verification & Validation

For purposes of brevity, Verification and Validation (V&V) are treated as a single topic in this Guide rather than as two separate topics as in the standard (IEEE12207.0-96). "Software V&V is a disciplined approach to assessing software products throughout the product life cycle. A V&V effort strives to ensure that quality is built into the software and that the software satisfies user requirements" (IEEE1059-93).

V&V addresses software product quality directly and uses testing techniques which can locate defects so that they can be addressed. It also assesses the intermediate products, however, and, in this capacity, the intermediate steps of the software life cycle processes.

The V&V process determines whether or not products of a given development or maintenance activity conform to the requirement of that activity, and whether or not the final software product fulfills its intended purpose and meets user requirements. Verification is an attempt to ensure that the product is built correctly, in the sense that the output products of an activity meet the specifications imposed on them in previous activities. Validation is an attempt to ensure that the right product is built, that is, the product fulfills its specific intended purpose. Both the verification process and the validation process begin early in the development or maintenance phase. They provide an examination of key product features in relation both to the product's immediate predecessor and to the specifications it must meet.

The purpose of planning V&V is to ensure that each resource, role, and responsibility is clearly assigned. The resulting V&V plan documents and describes the various resources and their roles and activities, as well as the techniques and tools to be used. An understanding of the different purposes of each V&V activity will help in the careful planning of the techniques and resources needed to fulfill their purposes.

The plan also addresses the management, communication, policies, and procedures of the V&V activities and their interaction, as well as defect reporting and documentation requirements.

3.3 Reviews and Audits

For purposes of brevity, reviews and audits are treated as a single topic in this Guide, rather than as two separate topics as in (IEEE12207.0-96). The review and audit process is broadly defined in (IEEE12207.0-96) and in more detail in (IEEE1028-97). Five types of reviews or audits are presented in the IEEE1028-97 standard:

- Management reviews
- Technical reviews
- Inspections
- Walk-throughs
- Audits

3.3.1 Management reviews

The purpose of a management review is to monitor progress, determine the status of plans and schedules, confirm requirements and their system allocation, or evaluate the effectiveness of management approaches used to achieve fitness for purpose. They support decisions about changes and corrective actions that are required during a software project. Management reviews determine the adequacy of plans, schedules, and requirements and monitor their progress or inconsistencies. These reviews may be performed on products such as audit reports, progress reports, V&V reports, and plans of many types, including risk management, project management, software configuration management, software safety, and risk assessment, among others.

3.3.2 Technical reviews

"The purpose of a technical review is to evaluate a software product to determine its suitability for its intended use. The objective is to identify discrepancies from approved specifications and standards. The results should provide management with evidence confirming (or not) that the product meets the specifications and adheres to standards, and that changes are controlled" (IEEE1028-97).

Specific roles must be established in a technical review: a decision-maker, a review leader, a recorder, and technical staff to support the review activities. A technical review requires that mandatory inputs be in place in order to proceed:

- Statement of objectives
- A specific software product
- The specific project management plan
- The issues list associated with this product
- The technical review procedure

The team follows the review procedure. A technically qualified individual presents an overview of the product, and the examination is conducted during one or more meetings. The technical review is completed once all the activities listed in the examination have been completed.

3.3.3 Inspections

The purpose of an inspection is to detect and identify software product anomalies. Two important differentiators of inspections as opposed to reviews are as follows:

- An individual holding a management position over any member of the inspection team shall not participate in the inspection.
- An inspection is to be led by an impartial facilitator who is trained in inspection techniques.

Software inspections always involve the author of an intermediate or final product, while other reviews might not. Inspections also include an inspection leader, a recorder, a reader, and a few (2 to 5) inspectors. The members of an inspection team may possess different expertise, such as domain expertise, design method expertise, or language expertise. Inspections are usually conducted on one relatively small section of the product at a time. Each team member must examine the software product and other review inputs prior to the review meeting, perhaps by applying an analytical technique to a small section of the product, or to the entire product with a focus only on one aspect, for example, interfaces. Any anomaly found is documented

and sent to the inspection leader. During the inspection, the inspection leader conducts the session and verifies that everyone has prepared for the inspection. A checklist, with anomalies and questions germane to the issues of interest, is a common tool used in inspections. The resulting list often classifies the anomalies and is reviewed for completeness and accuracy by the team. The inspection exit decision must correspond to one of the following three criteria:

- Accept with no or at most minor reworking
- Accept with rework verification
- Reinspect

Inspection meetings typically last a few hours, whereas technical reviews and audits are usually broader in scope and take longer.

3.3.4 Walk-throughs

The purpose of a walk-through is to evaluate a software product. A walk-through may be conducted for the purpose of educating an audience regarding a software product. The major objectives are to:

- Find anomalies
- Improve the software product
- Consider alternative implementations
- Evaluate conformance to standards and specifications

The walk-through is similar to an inspection but is typically conducted less formally. The walk-through is primarily organized by the software engineer to give his teammates the opportunity to review his work, as an assurance technique.

3.3.5 Audits

The purpose of a software audit is to provide an independent evaluation of the conformance of software products and processes to applicable regulations, standards, guidelines, plans, and procedures. The audit is a formally organized activity, with participants having specific roles, such as lead auditor, another auditor, a recorder, or an initiator, and includes a representative of the audited organization. The audit will identify instances of nonconformance and produce a report requiring the team to take corrective action.

While there may be many formal names for reviews and audits such as those identified in the standard (see IEEE1028- 97), the important point is that they can occur on almost any product at any stage of the development or maintenance process.

4 Practical Considerations

4.1 Software Quality Requirements

4.1.1 Influence factors

Various factors influence planning, management, and selection of SQM activities and techniques, including:

- The domain of the system in which the software will reside (safety-critical, mission-critical, business-critical)
- System and software requirements
- The commercial (external) or standard (internal) components to be used in the system
- The specific software engineering standards applicable
- The methods and software tools to be used for development and maintenance and for quality evaluation and improvement
- The budget, staff, project organization, plans, and scheduling of all the processes

- The intended users and use of the system
- The integrity level of the system

Information on these factors influences how the SQM processes are organized and documented, how specific SQM activities are selected, what resources are needed, and which will impose bounds on the efforts.

4.1.2 Dependability

In cases where system failure may have extremely severe consequences, overall dependability (hardware, software, and human) is the main quality requirement over and above basic functionality. Software dependability includes such characteristics as fault tolerance, safety, security, and usability. Reliability is also a criterion which can be defined in terms of dependability (ISO9126).

The body of literature for systems must be highly dependable ("high confidence" or "high integrity systems"). Terminology for traditional mechanical and electrical systems which may not include software has been imported for discussing threats or hazards, risks, system integrity, and related concepts, and may be found in the references cited for this section.

4.1.3 Integrity levels of software

The integrity level is determined based on the possible consequences of failure of the software and the probability of failure. For software in which safety or security is important, techniques such as hazard analysis for safety or threat analysis for security may be used to develop a planning activity which would identify where potential trouble spots lie. The failure history of similar software may also help in identifying which techniques will be most useful in detecting faults and assessing quality.

4.2 Defect Characterization

SQM processes find defects. Characterizing those defects leads to an understanding of the product, facilitates corrections to the process or the product, and informs project management or the customer of the status of the process or product. Many defect (fault) taxonomies exist, and, while attempts have been made to gain consensus on a fault and failure taxonomy, the literature indicates. Defect (anomaly) characterization is also used in audits and reviews, with the review leader often presenting a list of anomalies provided by team members for consideration at a review meeting.

As new design methods and languages evolve, along with advances in overall software technologies, new classes of defects appear, and a great deal of effort is required to interpret previously defined classes. When tracking defects, the software engineer is interested in not only the number of defects but also the types. Information alone, without some classification, is not really of any use in identifying the underlying causes of the defects, since specific types of problems need to be grouped together in order for determinations to be made about them. The point is to establish a defect taxonomy that is meaningful to the organization and to the software engineers.

SQM discovers information at all stages of software development and maintenance. Typically, where the word "defect" is used, it refers to a "fault" as defined below. However, different cultures and standards may use somewhat different meanings for these terms, which have led to attempts to define them. Partial definitions taken from standard (IEEE610.12-90) are:

- Error: "A difference...between a computed result and the correct result"
- Fault: "An incorrect step, process, or data definition in a computer program"
- Failure: "The [incorrect] result of a fault"
- Mistake: "A human action that produces an incorrect result"
- Failures found in testing as a result of software faults are included as defects in the discussion in this section. Reliability models are built from failure data collected during software testing or from software in service, and thus can be used to predict future failures and to assist in decisions on when to stop testing.

One probable action resulting from SQM findings is to remove the defects from the product under examination. Other actions enable the achievement of full value from the findings of SQM activities. These actions include analyzing and summarizing the findings, and using measurement techniques to improve the product and the process as well as to track the defects and their removal.

Data on the inadequacies and defects found during the implementation of SQM techniques may be lost unless they are recorded. For some techniques (for example, technical reviews, audits, inspections), recorders are present to set down such information, along with issues and decisions. When automated tools are used, the tool output may provide the defect information. Data about defects may be collected and recorded on an SCR (software change request) form and may subsequently be entered into some type of database, either manually or automatically, from an analysis tool. Reports about defects are provided to the management of the organization.

4.3 Software Quality Management Techniques

SQM techniques can be categorized in many ways: static, people-intensive, analytical, dynamic.

4.3.1 Static techniques

Static techniques involve examination of the project documentation and software, and other information about the software products, without executing them. These techniques may include people-intensive activities or analytical activities conducted by individuals, with or without the assistance of automated tools.

4.3.2 People-intensive techniques

The setting for people-intensive techniques, including reviews and audits, may vary from a formal meeting to an informal gathering or a desk-check situation, but (usually, at least) two or more people are involved. Preparation ahead of time may be necessary. Resources other than the items under examination may include checklists and results from analytical techniques and testing. These activities are discussed in (IEEE1028-97) on reviews and audits.

4.3.3 Analytical techniques

A software engineer generally applies analytical techniques. Sometimes several software engineers use the same technique, but each applies it to different parts of the product. Some techniques are tool-driven; others are manual. Some may find defects directly, but they are typically used to support other techniques. Some also include various assessments as part of overall quality analysis. Examples of such techniques include complexity analysis, control flow analysis, and algorithmic analysis.

Each type of analysis has a specific purpose, and not all types are applied to every project. An example of a support technique is complexity analysis, which is useful for determining whether or not the design or implementation is too complex to develop correctly, to test, or to maintain. The results of a complexity analysis may also be used in developing test cases. Defect-finding techniques, such as control flow analysis, may also be used to support another activity. For software with many algorithms, algorithmic analysis is important, especially when an incorrect algorithm could cause a catastrophic result. There are too many analytical techniques to list them all here. The list and references provided may offer insights into the selection of a technique, as well as suggestions for further reading.

Other, more formal, types of analytical techniques are known as formal methods. They are used to verify software requirements and designs. Proof of correctness applies to critical parts of software. They have mostly been used in the verification of crucial parts of critical systems, such as specific security and safety requirements.

4.3.4 Dynamic techniques

Different kinds of dynamic techniques are performed throughout the development and maintenance of software. Generally, these are testing techniques, but techniques such as simulation, model checking, and symbolic execution may be considered dynamic. Code reading is considered a static technique, but experienced software engineers may execute the code as they read through it. In this sense, code reading may also qualify as a dynamic technique. This discrepancy in categorizing indicates that people with different roles in the organization may consider and apply these techniques differently.

Some testing may thus be performed in the development process, SQA process, or V&V process, again depending on project organization. Because SQM plans address testing, this section includes some comments on testing.

4.3.5 Testing

The assurance processes described in SQA and V&V examine every output relative to the software requirement specification to ensure the output's traceability, consistency, completeness, correctness, and performance. This confirmation also includes the outputs of the development and maintenance processes, collecting, analyzing, and measuring the results. SQA ensures that appropriate types of tests are planned, developed, and implemented, and V&V develops test plans, strategies, cases, and procedures.

Two types of testing may fall under the headings SQA and V&V, because of their responsibility for the quality of the materials used in the project:

- Evaluation and test of tools to be used on the project (IEEE1462-98)
- Conformance test (or review of conformance test) of components and COTS products to be used in the product; there now exists a standard for software packages (IEEE1465-98)

Sometimes an independent V&V organization may be asked to monitor the test process and sometimes to witness the actual execution to ensure that it is conducted in accordance with specified procedures. Again, V&V may be called upon to evaluate the testing itself: adequacy of plans and procedures, and adequacy and accuracy of results.

Another type of testing that may fall under the heading of V&V organization is third-party testing. The third party is not the developer, nor is in any way associated with the development of the product. Instead, the third party is an independent facility, usually accredited by some body of authority. Their purpose is to test a product for conformance to a specific set of requirements.

4.4 Software Quality Measurement

The models of software product quality often include measures to determine the degree of each quality characteristic attained by the product.

If they are selected properly, measures can support software quality (among other aspects of the software life cycle processes) in multiple ways. They can help in the management decision-making process. They can find problematic areas and bottlenecks in the software process; and they can help the software engineers assess the quality of their work for SQA purposes and for longer-term process quality improvement.

With the increasing sophistication of software, questions of quality go beyond whether or not the software works to how well it achieves measurable quality goals.

There are a few more topics where measurement supports SQM directly. These include assistance in deciding when to stop testing. For this, reliability models and benchmarks, both using fault and failure data, are useful.

The cost of SQM processes is an issue which is almost always raised in deciding how a project should be organized. Often, generic models of cost are used, which are based on when a defect is found and how much effort it takes to fix the defect relative to finding the defect earlier in the development process. Project data may give a better picture of cost.

Finally, the SQM reports themselves provide valuable information not only on these processes, but also on how all the software life cycle processes can be improved.

While the measures for quality characteristics and product features may be useful in themselves (for example, the number of defective requirements or the proportion of defective requirements), mathematical and graphical techniques can be applied to aid in the interpretation of the measures. These fit into the following categories:

- Statistically based (for example, Pareto analysis, runcharts, scatter plots, normal distribution)
- Statistical tests (for example, the binomial test, chi-squared test)
- Trend analysis
- Prediction (for example, reliability models)

The statistically based techniques and tests often provide a snapshot of the more troublesome areas of the softwareproduct under examination. The resulting charts andgraphs are visualization aids which the decision-makerscan use to focus resources where they appear most needed. Results from trend analysis may indicate that a schedule has not been respected, such as in testing, or that certain classes of faults will become more intense unless some corrective action is taken in development. The predictive techniques assist in planning test time and in predicting failure.

References:

http://en.wikipedia.org/wiki/Software_quality_assurance, http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-171Fall2003/CourseHome/, http://www.cs.cornell.edu/courses/cs501/2008sp/, http://www.comp.l http://www.ee.unb.ca/kengleha/courses/CMPE3213/IntroToSoftwareEng.htm, http://www.softwarecertifications.org/qai_c http://satc.gsfc.nasa.gov/assure/agbsec3.txt, etc...